Theoretical Computer Science 60 (1988) 231-254
North-Holland

MATCHING PATTERNS IN STRINGS SUBJECT TO MULTI-LINEAR TRANSFORMATIONS

Tali EILAM-TZOREFF

Department of Computer Science, School of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel 69978

Uzi VISHKIN*

Department of Computer Science, School of Mathematical Sciences, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel 69978, and Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10003, U.S.A.

Communicated by M. Nivat
Received August 1987
Revised April 1988

Abstract. Suppose we are given two strings of real numbers. The longer string is called *text* and the other is called *pattern*. We consider problems within the following framework. Suppose each symbol of the pattern was modified by any transformation which is a member in some family of transformations. Find all occurrences of the pattern in the text where the pattern may appear subject to any one of these transformations. Problems are introduced and efficient algorithms are given.

Contents

1. Introduction

We recall the classical problem of matching patterns in strings considered in [1, 4, 6, 7, 15] and others. In this problem, we are given two strings: the pattern P

* The research of this author was supported by NSF Grants NSF-CCR-8615337 and NSF-DCR-8413359, ONR grant N00014-85-K-0046, by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract number DE-AC02-76ER03077 and by the Foundation for Research in Electronics, Computers and Communication, administered by the Israeli Academy of Sciences and Humanities. Present affiliation: The University of Maryland Institute for Advanced Computer Studies (UMIACS), College Park, MD 20742, U.S.A.

and the text T, and wish to find all (exact) occurrences of P in T. However, reality often requires finding close but not necessarily exact occurrences. The topic of nonexact string matching received considerable attention in the literature. For instance, [13] mentions nine papers in three fields (molecular biology, speech recognition and computer science), which gave independently the same dynamic programming algorithm for finding the number of insertions, deletions and substitutions required to transform one given string into another given string (the String Edit problem). Recently, in [9, 12] the problem of matching patterns in strings is considered where a bounded number of such edit operations between the text and the pattern is allowed.

In the present paper we introduce extensions of the exact string matching problem in another "dimension". We consider several problems. In each problem we are given a family of transformations that can be applied to the pattern. We then search for occurrences of the pattern in the text, where in each occurrence the pattern may appear subject to *any one* of the allowed transformations. The description given so far dictates a simple approach where each search for occurrence of the pattern in some location of the text does not necessarily benefit from previous searches with respect to other locations. We sometimes refer to this as the naive approach. However, the following challenge was met by the designers of most of the above string matching algorithms: to identify features of the specific pattern at hand for deducing leads regarding occurrences at neighboring locations. The main effort in this research was devoted to exploring a similar challenge within the context of our problems.

The input to all problems in the present paper consists of the pattern P and the text T. We denote $|P|$ (the length of P) by m, and $|T|$ by n. Each element in the text or in the pattern is a *real number.*

We consider the following problems:

(1) *The adding transformation problem*: For each j, $1 \leq j \leq n-m+1$, find whether there exists a constant c_0 such that $\forall i$, $1 \leq i \leq m$, $T_{j-1+i} = P_i + c_0$. A match is called an *adding transformation* occurrence. Note that we may find different adding transformations for different occurrences.

Example. Let the text be $T = 1, 4, 5, 6, 8, 9, 10$ and the pattern $P = 1, 2, 3$. There are two adding transformation occurrences of the pattern in the text. One starts at the second position of the text with $c_0 = 3$ and the other starts at the fifth position with $c_0 = 7$.

(2) *The multiplying transformation problem*: For each j, $1 \leq j \leq n-m+1$, find whether there exists a constant c_1 such that $\forall i$, $1 \leq i \leq m$, $T_{j-1+i} = c_1 P_i$. A match is called a *multiplying transformation* occurrence.

(3) *The linear transformation problem*: For each j, $1 \leq j \leq n-m+1$, find whether there exist two constants c_0, c_1 such that $\forall i$, $1 \leq i \leq m$, $T_{j-1+i} = c_1 P_i + c_0$. A match is called a *linear transformation* occurrence.

(4) *The k-degree polynomial transformation problem*: For each j, $1 \leq j \leq n-m+1$, find whether there exist $k+1$ constants $c_0, c_1, \ldots, c_k$ such that $\forall i$, $1 \leq i \leq m$, $T_{j-1+i} = \sum_{l=0}^{k} c_l P_i^l$ (where P_i^l is the ith element of the pattern P raised to the lth power). A match is called a *k-degree polynomial transformation* occurrence.

(5) *The k-linear transformation problem*: We are given k patterns $B_1, \ldots, B_k$. For each j, $1 \leq j \leq n-m+1$, find whether there exist k constants $c_1, \ldots, c_k$ such that $\forall i$, $1 \leq i \leq m$, $T_{j-1+i} = \sum_{l=1}^{k} c_l B_{l,i}$ (where $B_{l,i}$ denotes the ith element of the pattern B_l). A match is called a *k-linear transformation* occurrence of the k patterns $B_1, \ldots, B_k$ in the text.

The rest of this introduction outlines relations among these five problems, where elementary considerations and solutions suffice. Later, we overview the main results of the paper and provide some more motivation.

The algorithm for the adding transformation problem is trivial. We provide a reduction of the adding transformation problem into the exact string matching problem. Instead of the given text T we look at $\bar{T}$, the *sequence of differences* of T, which is defined as follows:

$$\bar{T}_j = T_{j+1} - T_j \quad 1 \leq j \leq n-1.$$

Similarly, instead of P we look at $\bar{P}$; its sequence of differences, where

$$\bar{P}_i = P_{i+1} - P_i \quad 1 \leq i \leq m-1.$$

The reduction is based on the following observation. An adding transformation occurrence of P starts at position j in T if and only if an (exact) occurrence of $\bar{P}$ starts at position j in $\bar{T}$. Thus, we can apply any of the known linear-time algorithms for the exact string-matching problem for $\bar{T}$ and $\bar{P}$. The constant c_0 for each position j (in which there is an occurrence) is determined by solving the equation $T_j = P_1 + c_0$.

We provide a reduction of the multiplying transformation problem into the exact string-matching problem. Let $\bar{T}$ and $\bar{P}$ be the *sequences of quotients* of T and P, respectively, which are defined as follows: $\bar{T}_j = T_{j+1}/T_j$ for $1 \leq j \leq n-1$, and $\bar{P}_i = P_{i+1}/P_i$ for $1 \leq i \leq m-1$ (assuming that neither the text nor the pattern contain zeros). The reduction is based on the following observation. A multiplying transformation occurrence of P starts at position j in T if and only if an (exact) occurrence of $\bar{P}$ starts at position j in $\bar{T}$. Again, we can apply a linear-time algorithm for the exact string-matching problem for $\bar{T}$ and $\bar{P}$. The constant c_1 for each position j (in which there is an occurrence) is determined by solving the equation $T_j = c_1 P_1$.

We provide a reduction of the linear transformation problem into the multiplying transformation problem. This enables us to consider only the latter problem later in this paper. Let $\bar{T}$ and $\bar{P}$ be the sequences of differences of the text and the pattern respectively. The reduction is based on the following observation. A linear transformation occurrence of P starts at position j in T if and only if a multiplying transformation occurrence of $\bar{P}$ starts at position j in $\bar{T}$. So we apply the multiplying transformation algorithm for $\bar{T}$ and $\bar{P}$. The constant c_0 for each position j (in which

there is an occurrence) is determined by solving the equation $T_j = c_1 P_1 + c_0$ (where the constant c_1 was computed by the multiplying transformation algorithm).

The k-degree polynomial transformation problem is included in the $(k+1)$-linear transformation problem. To see this, simply choose the pattern $P_1^l P_2^l \ldots P_m^l$ (denoted P^l) as B_l, $1 \leq l \leq k$, and the pattern consisting of m ones (denoted P^0) as B_{k+1}. Therefore, we elaborate only on the k-linear transformation problem.

We give algorithms for the multiplying transformation problem and the k-linear transformation problem. Each of these two algorithms consists of two steps. First some table is computed based on analysis of the pattern. Second, the text is analysed in order to solve the respective problem. In the multiplying transformation algorithm the analysis of the pattern takes $O(m)$ (linear) time and the analysis of the text $O(n)$ (linear) time. An alternative algorithm which is simpler is described in Remark 2.6. However, we preferred to elaborate on a multiplying transformation algorithm whose text analysis has the advantage of being smoothly generalizable into a k-linear transformation algorithm. In the k-linear transformation algorithm the analysis of the pattern takes $O(k^2m^2)$ time and the analysis of the text $\min\{O(kmn), O(k^3n)\}$ time.

We mentioned five problems which are considered in this paper. Their formulation is motivated by problems which arise in fields like speech recognition and image processing, where human speech or images are represented as arrays of real numbers. In speech recognition an utterance, in the form of a continuous multidimensional function of time, is converted into a sequence of points in multidimensional space by sampling at regular time intervals.

The general setting of the problems considered here is as follows. We get some complex string (one-dimensional array) and one (or several) more basic string. The basic string may occur in the complex string subject to a single transformation which had been selected arbitrarily from a whole family of possible transformations. Actually, the basic string may occur several times in the same complex string, each time subject to a different transformation. We know in advance the family of transformations, but do not know which transformations have actually been used. The problem is to find all occurrences of the basic string in the complex string and their respective transformations. In practice, the demand of finding an exact match between the pattern (following any of the transformations mentioned above) and the text might be too strict since real numbers are likely to occur with variants. Therefore, we define for each of these five (exact) problems a minimum distance problem. We give here only the precise definition for the minimum distance k-linear problem. We define the minimum distance between the pattern subject to k-linear transformation and the text starting at position j, as follows. For each position j, find k numbers $c_{j,1}, c_{j,2}, \ldots, c_{j,k}$ which provide the minimum in

$$L_j = \sum_{i=1}^{m} (T_{j-1+i} - \sum_{l=1}^{k} c_{j,l} B_{l,i})^2.$$

The minimum distance k-linear transformation problem is to find a position j and numbers $c_{j,1}, \ldots, c_{j,k}$ for which L_j is a global minimum.

We give an algorithm for finding a minimum distance occurrence of the pattern (without any transformation) in the text. The algorithm runs in $\min\{O(nm), O(n \log n)\}$ time. We also present algorithms for minimum distance versions of the above transformation problems. The minimum distance algorithms for adding transformation, and linear transformation run in $\min\{O(nm), O(n \log n)\}$ time each. The minimum distance algorithm for k-linear transformation runs in $\min\{O(k(k+m)n), O(k(k+\log n)n)\}$ time.

In Section 2, we present an algorithm for the multiplying transformation problem. In Section 3, we present an algorithm for the k-linear transformation problem. In Section 4, we present the minimum distance algorithms. Technically, the more interesting parts of the paper are Section 3 and the minimum distance k-linear algorithm of Section 4. Remarks 2.7 and 4.4 on parallel algorithms can be found in Sections 2 and 4.

Note added in proof. Recently, Vishkin and Yedidia [16] have parallelized the text analysis part of the k-linear transformation problem. Their parallel text analysis algorithm runs in $O(k^2 \log n)$ time using n processors.

2. Algorithm for the multiplying transformation problem

The algorithm has two stages: (1) analysis of the pattern; (2) analysis of the text. In this section, we describe the text analysis first and the pattern analysis later.

2.1. Text analysis

The input for the text analysis is a table, called *witness*, which is computed in the pattern analysis. Let us define this table. For each j, $2 \leq j < m$, *witness*(j) is i if there exists a position i, $1 < i \leq m-j+1$ such that $P_1/P_j \neq P_i/P_{j-1+i}$; and *witness*$(j)$ is ∞ if no such i exists (cf. Fig. 1).

The text analysis consists of three steps. At the beginning, each position j in the text, $1 < j \leq n-m+1$, is considered a candidate for being the start of a multiplying transformation occurrence of the pattern. The first step determines for each position j a constant c_j such that if a multiplying transformation starts at j, then the multiplier must be c_j. In the second step, the candidacy of some of the positions in the text

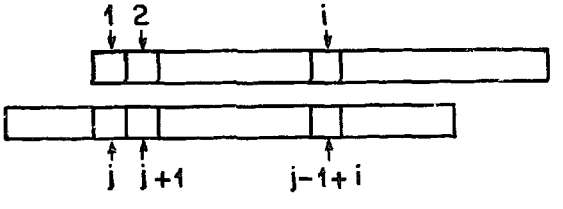

Fig. 1.

is invalidated. The third step applies a kind of character-by-character check to finally determine in which of the remaining candidates an occurrence of the pattern really starts. A detailed description of the text analysis follows.

Step 1

For each position j in the text we fix c_j by solving the equation $T_j = c_j P_1$.

We describe a mechanism, called "duel", for the purpose of invalidating candidacy of positions in the text. Step 2, given later, consists of applying this mechanism. Let $j_2 > j_1$ be two positions in the text which are less than m apart (i.e., $j_2 - j_1 < m$). Suppose that, following Step 1, j_1 and j_2 are candidates for the occurrences of $c_{j_1}P$ and $c_{j_2}P$, respectively.

The idea of duel between j_1, j_2. Suppose $witness(j_2 - j_1 + 1) \neq \infty$, and denote $witness(j_2 - j_1 + 1)$ by i. We show how to invalidate the candidacy of at least one of the two positions j_1, j_2 (cf. Fig. 2). We know that

$$T_{j_2} = c_{j_2} P_1, \qquad T_{j_1} = c_{j_1} P_1. \tag{2.1}$$

Instruction 1 *of the duel*: Check whether the following equality holds:

$$T_{j_2} = c_{j_1} P_{j_2 - j_1 + 1}. \tag{2.2}$$

If not, then the candidacy of j_1 is invalidated. If the equality holds, then, by the definition of the *witness* table, we have

$$\frac{P_1}{P_{j_2 - j_1 + 1}} \neq \frac{P_i}{P_{j_2 - j_1 + i}}. \tag{2.3}$$

The following observation is essential for the duel idea.

Observation. *At most one of the following two equalities may hold*:

$$T_{j_2 - 1 + i} = c_{j_2} P_i, \tag{2.4}$$

$$T_{j_2 - 1 + i} = c_{j_1} P_{j_2 - j_1 + i}. \tag{2.5}$$

Proof. Assume to the contrary that both equalities hold. This implies

$$\frac{P_1}{P_{j_2 - j_1 + 1}} = \frac{T_{j_2}/c_{j_2}}{T_{j_2}/c_{j_1}} = \frac{T_{j_2 - 1 + i}/c_{j_2}}{T_{j_2 - 1 + i}/c_{j_1}} = \frac{P_i}{P_{j_2 - j_1 + i}}.$$

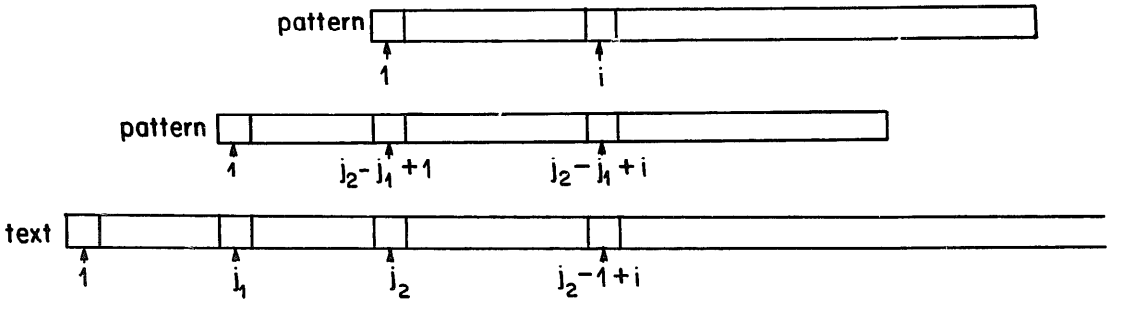

Fig. 2.

The first equality is by (2.1) and (2.2). The second is trivial. The third is by (2.4) and (2.5). This contradicts inequality (2.3) and the observation follows. □

The duel proceeds as follows.

Instruction 2: If $T_{j_2-1+i} \neq c_{j_2} P_i$, then the candidacy of j_2 is invalidated, and if $T_{j_2-1+i} \neq c_{j_1} P_{j_2-j_1+i}$, then the candidacy of j_1 is invalidated. We say that an invalidated candidate is a *loser* in the duel (note that it is possible that both participants in a duel are losers).

If $witness(j_2-j_1+1)=\infty$, then

$$\frac{P_1}{P_{j_2-j_1+1}}=\frac{P_i}{P_{j_2-j_1+i}} \quad \text{for all } i,\ 1<i\leq m-j_2+j_1.$$

We still perform instruction 1. Instruction 2 is not performed since for each i both equations (2.4) and (2.5) may hold. Thus, it is possible that both j_1 and j_2 remain valid following the duel.

2.1. Remark. Notions similar to "witness" and "duel" were used in the parallel (exact) string matching algorithm of [15].

Step 2

At the beginning of this step the first $n-m+1$ positions of the text, which are all candidates for multiplying transformation occurrences, form a (doubly linked) list. The processing of the list proceeds in *substeps*. Each substep consists (essentially) of performing a duel between two adjacent elements in the list. Let us be more specific. The first substep performs a duel between positions $n-m+1$ and $n-m$ in the text. A typical substep proceeds as follows. The substep consists (essentially) of a duel between two elements B and C in the list. Suppose that C is the successor of B in the current list, where B represents position j and C represents position k in the text (note that $j>k$). Any element representing a loser in the duel of the present substep is deleted from the list. The principle for determining the two elements which participate in the next substep is as follows. If one or two elements were deleted in the present substep, then two elements (which previously were not adjacent) become adjacent. These two elements are the participants in the next substep. Let D be the successor of C in the list (if it exists). If no element was deleted, then C and D are the participants in the next substep. The termination condition of Step 2 is as follows: D does not exist and had D existed it would have participated in the next substep. When Step 2 is over it is impossible to perform an "effective" duel (i.e., a duel in which at least one element loses) between any pair of adjacent elements in the list. (For a more precise statement see Fact 2.2 below). We proceed to a more detailed description of Step 2. Let A be the predecessor of B. For simplicity we assume that A exists. (It is simple to extend the definition of a substep where this assumption is false.)

(a) If j and k are at least m apart (i.e., $j-k \geq m$), then a duel is not performed. The participants in the next substep are the elements C and D in the list. Step 2 terminates if D does not exist.

(b) If j and k are less than m apart, then a duel is performed between them. In order to determine the participants of the next substep we distinguish between four cases:

(1) There were no losers in the duel. The participants in the next substep are C and D. Step 2 terminates if D does not exist.

(2) Both participants in the duel were losers. We delete the elements B and C from the list. The participants in the next substep are A and D. Step 2 terminates if D does not exist.

(3) Only C lost in the duel. We delete C from the list. The participants in the next substep are B and D. Step 2 terminates if D does not exist.

(4) Only B lost in the duel. We delete B from the list. The participants in the next substep are A and C.

The following fact and corollary will be used in the description of Step 3.

2.2. Fact. *Consider the end of Step 2. Let X be an element of the list and Y its successor. Then there must have been a substep of Step 2 in which elements X and Y participated.*

Proof. Let i be any substep of Step 2. Let C and B be the two participants of substep i where C is the successor of B in the list. The proof of Fact 2.2 proceeds by induction on i. The inductive step assumes that every pair of successive elements in the portion of the list which starts at the beginning of the list and ends at element B before substep i begins, has participated in a substep. It is left to the reader to verify that, following substep i, every pair of successive elements in the portion of the list, which starts at the beginning of the list and ends at the first of two elements which participate in substep $i+1$ (or "end of list" if there is no substep $i+1$), has participated in a substep. Fact 2.2 follows. □

2.3. Corollary. *Consider the end of Step 2. Let j_1 and j_2 $(j_2 > j_1)$ be any two adjacent elements in the list. Then, one of the two following conditions holds:*

(1) $j_2 - j_1 \geq m$;

(2) $j_2 - j_1 < m$ *and* $\forall i,\ 1 < i \leq m-(j_2-j_1)$, $c_{j_2}P_i = c_{j_1}P_{j_2-j_1+i}$.

Proof. Let j_1 and j_2 be two elements from the list as in Fact 2.2. If $j_2 - j_1 \geq m$, then condition (1) holds. Suppose $j_2 - j_1 < m$. By Fact 2.2 we know that in Step 2 a duel was performed between them. No one has lost. Thus, (see Fig. 2)

$$c_{j_2}P_1 = T_{j_2} = c_{j_1}P_{j_2-j_1+1}$$

by the definition of *witness*

$$\frac{c_{j_1}}{c_{j_2}} = \frac{P_1}{P_{j_2-j_1+1}} = \frac{P_i}{P_{j_2-j_1+}} \quad \forall i,\ 1 < i \leq m-(j_2-j_1).$$

Therefore,

$$c_{j_2}P_i = c_{j_1}P_{j_2-j_1+i} \quad \forall i, 1 < i \leq m-(j_2-j_1). \qquad \square$$

We say that any two positions j_1 and j_2 ($j_2 > j_1$) in the text which satisfy Corollary 2.3 *agree*. Corollary 2.3 shows that any two adjacent elements in the list at the end of Step 2 agree.

2.4. Remark. Although not essential for the description of Step 3, it is interesting to notice that the following transitivity property can be proven: Every two elements in the list at the end of Step 2 (not necessarily adjacent) do agree.

Step 3

Let l be the number of elements in the list following Step 2. Step 3 proceeds in l substeps. In substep i, we finish verifying whether a multiplying transformation occurrence starts at (the position in the text of) element i in the list, where substep 1 verifies such an occurrence at the element of the list representing the highest index in the text. We describe a typical substep i. Suppose element i in the list represents index j_i in the text. There are three basic possibilities regarding relevant information from the previous substep.

Possibility 1: the indices j_{i-1} and j_i of the text are less than m apart (i.e., $j_{i-1}-j_i < m$) and there is a multiplying occurrence at index j_{i-1} of the text. Denote by k_i the integer satisfying $j_{i-1} = j_i - 1 + k_i$ ($k_i \leq m$). By Corollary 2.3, $T_{j_i-1+k} = c_{j_i}P_k$ for every $k_i \leq k \leq m$. It remains to check whether $T_{j_i-1+k} = c_{j_i}P_k$ for all $2 \leq k < k_i$. This is done in (at most) $k_i - 2$ checks for decreasing values of k. If all $k_i - 2$ checks show equality, we conclude that there is an occurrence at index j_i and proceed to substep $i+1$. If in one of these checks we find that $T_{j_i-1+k} \neq c_{j_i}P_k$, then we conclude that there is no occurrence at index j_i and proceed to substep $i+1$.

Possibility 2: $j_{i-1} - j_i < m$ and there is no multiplying occurrence at index j_{i-1} of the text. Specifically, we assume that substep $i-1$ found $k_{i-1} \leq m$ such that $T_{j_{i-1}-1+k_{i-1}} \neq c_{j_{i-1}}P_{k_{i-1}}$.

(a) If $j_{i-1} - 1 + k_{i-1} \leq j_i - 1 + m$, then denote by k_i the integer satisfying $j_{i-1} - 1 + k_{i-1} = j_i - 1 + k_i$. Clearly, $1 < k_i \leq m$. By Corollary 2.3 we get that $T_{j_i-1+k_i} \neq c_{j_i}P_{k_i}$. We conclude that there is no multiplying occurrence at j_i and proceed to substep $i+1$ (cf. Fig. 3).

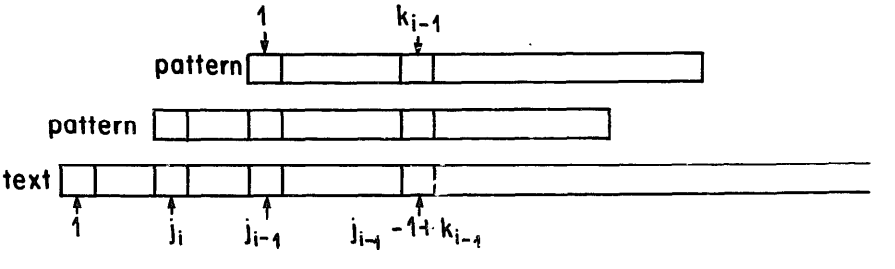

Fig. 3.

(b) If $j_{i-1}-1+k_{i-1}>j_i-1+m$, then we have to check whether $T_{j_{i-1}+k}=c_{j_i}P_k$ for all $2\leq k\leq m$. Thus, we initialize $k_i=m+1$ and apply Possibility 1.

Possibility 3: the indices j_{i-1} and j_i of the text are at least m apart (i.e., $j_{i-1}-j_i\geq m$). Again, we have to check whether $T_{j_{i-1}+k}=c_{j_i}P_k$ for all $2\leq k\leq m$, so we initialize $k_i=m+1$ and apply Possibility 1.

Correctness. At the end of Step 3 we get a list of elements representing all positions in the text in which a multiplying transformation occurrence of the pattern starts.

Time complexity

Step 1: For each position j, c_j is determined in O(1) time. Therefore, Step 1 needs $O(n)$ time.

Step 2: We show that in Step 2 we perform $O(n)$ substeps, each takes O(1) time. Thus, Step 2 needs $O(n)$ time. In each substep one of the following must occur:

(i) There is a "new" participant in the substep (i.e., an element which did not participate in any former substep). At most $n-m+1$ such substeps are possible since this is the initial number of elements in the (doubly linked) list.

(ii) An element was deleted from the list. At most $n-m+1$ such substeps are possible.

(iii) Neither case (i) nor case (ii) applies. That is, there was no new participant in the substep and no element was deleted from the list. Such a substep must be preceded by a substep which falls in case (b)(4) of the detailed description of Step 2. Since case (b)(4) results in deleting an element we "charge" the substep to this deleted element. Therefore, at most $n-m+1$ such substeps are possible.

We showed that Step 2 has no more than $3(n-m+1)$ substeps.

Step 3: Each position in the text is compared to a pattern element at most once. Therefore, Step 3 needs $O(n)$ time.

We conclude that the text analysis takes a total of $O(n)$ time.

2.2. *Pattern analysis*

We compute the *witness* table as defined at the beginning of this section. Actually, our algorithm will be slightly stronger than required: *witness*(j) will be i only if i is the *smallest* index such that $P_1/P_j\neq P_i/P_{j-1+i}$. The algorithm proceeds in cycles. We first describe the first cycle and then a general cycle which follows the principles of the first cycle but is more involved.

The first cycle

The first cycle consists of a few steps. In the first step *witness*(2) is being computed. The first step proceeds by checking whether $P_1/P_2=P_{last-1}/P_{last}$ for $last=2,3,\ldots$ until either (Possibility 1) the first time equality does not hold or (Possibility 2) $last=m+1$ (cf. Fig. 4).

Possibility 1: *last* is the smallest index for which $P_1/P_2\neq P_{last-1}/P_{last}$. We conclude that *witness*(2) $=last-1$. The first cycle has $last-3$ additional steps. In step $k-1$, $2<k<last$, we fix *witness*(k) to be $last-(k-1)$ (see Fig. 4).

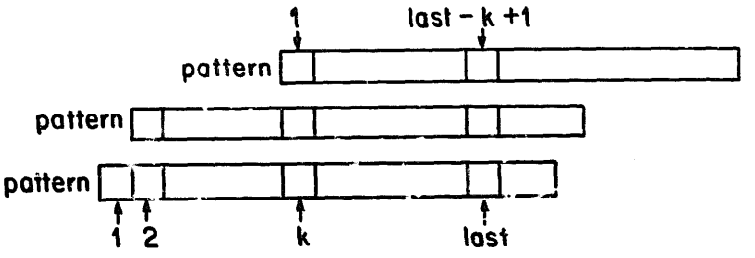

Fig. 4.

Possibility 2: $last = m+1$. Then $witness(2) = \infty$ and we conclude that $witness(k) = \infty$ for all $2 < k \leq m$. The whole algorithm is finished within the first cycle.

A general cycle

In the first step we compute $witness(j-1)$ for some $j > 2$. Inductively, the previous cycle guarantees that for some $L \geq j-1$ the equality $P_1/P_{j-1} = P_{last-j+2}/P_{last}$ holds for every $j-1 \leq last \leq L$. The first step proceeds by checking whether this equality holds for $last = L+1, L+2, \ldots$ until the first time this equality does not hold (Possibility 1) or $last = m+1$ (Possibility 2).

Possibility 1: *last* is the smallest index for which $P_1/P_{j-1} \neq P_{last-j+2}/P_{last}$. We set $witness(j-1) = last-j+2$. The cycle has at most $last-j$ additional steps. Description of a typical step follows. In step $k-j+2$, $j-1 < k < last$, we compute $witness(k)$. Denote $i = k-j+2$. We distinguish among three cases. In Cases 1 and 2 below, the value of $witness(k)$ is derived from $witness(i)$ only, while for Case 3 we need more comparisons.

Case 1: $(k + witness(i) - 1) < last$ (cf. Fig. 5). Since $witness(j-1) = last-j+2$,

$$\frac{P_1}{P_{j-1}} = \cdots = \frac{P_i}{P_k} = \frac{P_{i+1}}{P_{k+1}} = \cdots = \frac{P_{i+witness(i)-2}}{P_{k+witness(i)-2}} = \frac{P_{i+witness(i)-1}}{P_{k+witness(i)-1}}.$$

A later remark implies that $witness(i) \neq \infty$, and therefore,

$$\frac{P_1}{P_i} = \frac{P_2}{P_{i+1}} = \cdots = \frac{P_{witness(i)-1}}{P_{i+witness(i)-2}} \neq \frac{P_{witness(i)}}{P_{i+witness(i)-1}}.$$

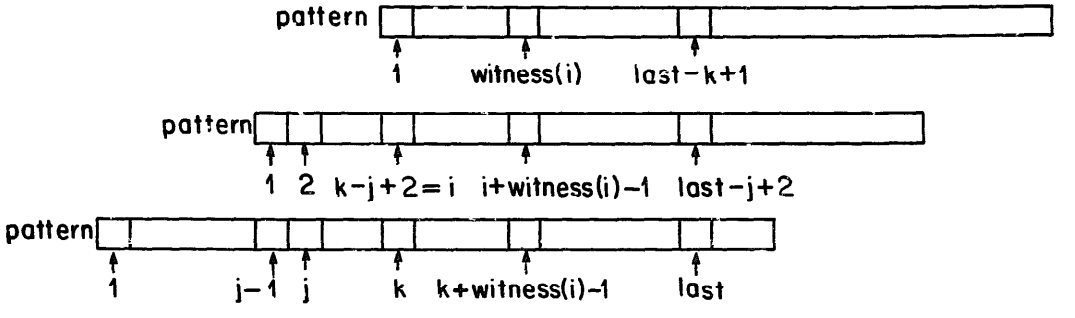

Fig. 5.

By multiplying we get,

$$\frac{P_1}{P_k}=\frac{P_2}{P_{k+1}}=\cdots=\frac{P_{witness(i)-1}}{P_{k+witness(i)-2}}\neq\frac{P_{witness(i)}}{P_{k+witness(i)-1}}.$$

We conclude that $witness(k)=witness(i)$. If this is the last step in the present cycle (i.e., $k=last-1$), then we proceed to the next cycle, where each of L and $j-1$ gets the value *last*.

Case 2: $(k+witness(i)-1)>last$. We know that

$$\frac{P_1}{P_{j-1}}=\cdots=\frac{P_i}{P_k}=\frac{P_{i+1}}{P_{k+1}}=\cdots=\frac{P_{last-j+1}}{P_{last-1}}\neq\frac{P_{last-j+2}}{P_{last}},$$

$$\frac{P_1}{P_i}=\frac{P_2}{P_{i+1}}=\cdots=\frac{P_{last-k}}{P_{last-j+1}}=\frac{P_{last-k+1}}{P_{last-j+2}}$$

which implies that

$$\frac{P_1}{P_k}=\frac{P_2}{P_{k+1}}=\cdots=\frac{P_{last-k}}{P_{last-1}}\neq\frac{P_{last-k+1}}{P_{last}}.$$

We conclude that $witness(k)=last-k+1$. If this is the last step in the present cycle, then we proceed to the next cycle, as in Case 1.

Case 3: $(k+witness(i)-1)=last$. We know that

$$\frac{P_1}{P_{j-1}}=\cdots=\frac{P_i}{P_k}=\frac{P_{i+1}}{P_{k+1}}=\cdots=\frac{P_{last-j+1}}{P_{last-1}}\neq\frac{P_{last-j+2}}{P_{last}},$$

$$\frac{P_1}{P_i}=\frac{P_2}{P_{i+1}}=\cdots=\frac{P_{last-k}}{P_{last-j+1}}\neq\frac{P_{last-k+1}}{P_{last-j+2}}$$

which implies that

$$\frac{P_1}{P_k}=\frac{P_2}{P_{k+1}}=\cdots=\frac{P_{last-k}}{P_{last-1}}.$$

However, we cannot know (without actually comparing) whether P_1/P_k equals $P_{last-k+1}/P_{last}$. In this case, the cycle ends, $j-1$ gets the value k, L gets the value $last-1$, and we proceed to the next cycle.

2.5. Remark. Note that the case $witness(i)=\infty$ is impossible since, in order to find that $witness(i)=\infty$, we would have exhausted the pattern, but $last\leq m$.

Possibility 2: $last=m+1$. Then $witness(j-1)=\infty$. In this case, the whole algorithm is finished within this cycle. Again, in step $k-j+2$, $j-1<k<last$, we compute $witness(k)$, and denote $i=k-j+2$. We consider two cases.

Case 1: $(k+witness(i)-1)<last$. Apply Case 1 of Possibility 1.

Case 2: $(k+witness(i)-1) \geq last$. We know that

$$\frac{P_1}{P_{j-1}} = \cdots = \frac{P_i}{P_k} = \frac{P_{i+1}}{P_{k+1}} = \cdots = \frac{P_{m-j+2}}{P_m},$$

$$\frac{P_1}{P_i} = \frac{P_2}{P_{i+1}} = \cdots = \frac{P_{m-k+1}}{P_{m-j+2}}.$$

We get

$$\frac{P_1}{P_k} = \frac{P_2}{P_{k+1}} = \cdots = \frac{P_{m-k+1}}{P_m}.$$

We conclude that $witness(k) = \infty$.

Time complexity

We charge each operations performed by the algorithm to an element of the pattern. We show that we charge at most a constant number of operations to each element. This implies that the pattern analysis takes a total of $O(m)$ time.

Consider the first step of a general cycle. P_1/P_{j-1} is compared to $P_{last-j+2}/P_{last}$. If equality holds, we charge P_{last}, and if not, we charge P_{j-1}. Summing up over the first step of all cycles, each element of the pattern will be charged at most once due to equality and at most once due to inequality. Each non-first step needs a constant number of operations. For each element k of the pattern, the computation of $witness(k)$ is considered in at most one non-first step. Therefore, both first and non-first steps of all cycles take $O(m)$ time.

2.6. Remark. Consider a stage i in the text analysis of the exact string matching algorithm of [6]. Such a stage obeys the following framework. Suppose stage $i-1$ searched for a match of the pattern starting at position j_{i-1}. Suppose positions $j_{i-1}, j_{i-1}+1, \ldots, j_{i-1}+x$ of the text have been examined by the end of stage $i-1$. A "failure function" (computed in the pattern analysis), which depends only on x, will provide a position j_i. Step i will search a match of the pattern starting at position j_i without re-examining positions of the text. This framework can be adapted for the multiplying transformation problem. However, our algorithm above is somewhat more involved so that it will be smoothly generalizable for the k-linear transformation problem. Specifically, the difficulty is the following. Had we followed the same framework, the determination of position j_i for stage i of the k-linear algorithm may have to depend on some stage prior to stage $i-1$.

2.7. Remark. The optimal parallel pattern matching algorithm of [15] can be adapted into an optimal parallel algorithm for the multiplying transformation problem. We mention the main changes and leave the details to the interested reader. We change the definition of the term *period* as follows. Suppose u, w two strings of real numbers are given. We say that u is a period of w if there exists a constant c

such that w is a prefix of $u(cu)(c^2u)\ldots(c^ku)$ for some k. (c^iu, $1 \leq i \leq k$, is the string obtained by multiplying each element of u by c^i). Note that this definition permits the proof of a "Periodicity Lemma" (similar to the Periodicity Lemma in [15]). The definition of the *witness* table is the same as in the present section. We note two additional things.

(1) Suppose a string u is a period of the pattern P. Let $q = |u|$. Then

$$\frac{P_1}{P_{q+1}} = \frac{P_2}{P_{q+2}} = \cdots = \frac{P_{m-q}}{P_m}$$

and therefore $witness(q+1) = \infty$.

(2) Throughout the whole algorithm of [15] we automatically replace each check of the form $P_i \neq P_j$ (where $i < j$) by the check $P_1/P_{j-i+1} = P_i/P_j$.

3. Algorithm for the k-linear transformation problem

This algorithm has two stages: (1) analysis of the pattern; (2) analysis of the text. Some other characteristics of the algorithm will also be similar to the algorithm of the previous section. The output of the pattern analysis is a table called *witness*, which is defined as follows.

Figure 6 describes the k input patterns $B_1, \ldots, B_k$, all starting at the same vertical location and again these same k patterns starting at location j of the former k patterns. (Following the figure, we call the former k patterns *lower* and the latter k patterns *upper*.) Figure 6 (and our present definition of $witness(j)$) will attest to the coexistence of the k patterns at two locations of the text which are $j-1$ apart. Each location l in the upper patterns *suggests* the following equation with $2k$ unknowns $c_1, c_2, \ldots, c_{2k}$:

$$\sum_{i=1}^{k} c_i B_{i,l} = \sum_{i=1}^{k} c_{k+i} B_{i,j+l-1}.$$

The equations at all locations $1 \leq l \leq m-j+1$, capture exactly the degree of freedom for such coexistence (subject to the k-linear rule).

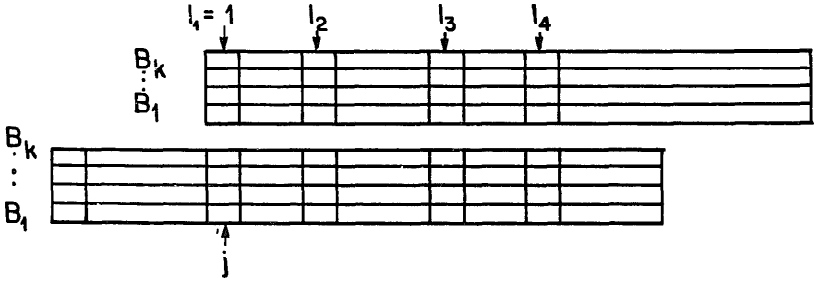

Fig. 6.

For each j, $2 \leq j < m$, *witness*(j) is a $2k$-tuple $(l_1, \ldots, l_{2k})$. *witness*(j) will include r locations in the upper patterns. These r locations suggest the maximum number of linearly independent equations. (Clearly, r is at most $2k$). We determine these r locations as follows. l_1 is 1. Once the $i-1$ independent equations of locations $l_1, \ldots, l_{i-1}$ are given, l_i is the least location, in the upper patterns, which gives the ith independent equation. (If an ith independent equation does not exist, $l_i, \ldots, l_{2k}$ are all zero.) We describe the text analysis first and the pattern analysis later.

3.1. Text analysis

The text analysis consists of three steps. In this section S_j will represent a set whose elements are k-tuples of real numbers. The first step initializes for each position j (in the text) a set S_j such that if a k-linear transformation occurrence starts at j, then the k multipliers must be a k-tuple $c_{j,1}, \ldots, c_{j,k}$ in S_j. In the second step, the candidacy of some positions in the text is invalidated. The third step applies a kind of character-by-character check to finally find in which of the remaining candidates a k-linear transformation occurrence of the pattern really starts. A detailed description of the text analysis follows.

Step 1

For each position j in the text we initialize S_j by attempting to find a k-linear match to an easier problem. Specifically, we attempt to match a prefix of length k of the k patterns to the substring of length k starting at T_j. For this we solve the following linear system with $c_{j,i}$, $1 \leq i \leq k$, as unknowns:

$$\sum_{i=1}^{k} c_{j,i} B_{i,1} = T_j, \quad \sum_{i=1}^{k} c_{j,i} B_{i,2} = T_{j+1}, \ldots, \sum_{i=1}^{k} c_{j,i} B_{i,k} = T_{j+k-1}.$$

If S_j is empty (i.e., this system does not have any solution), then a k-linear occurrence at T_j is impossible and we can already invalidate the candidacy of j. (Observe that this is unlike the multiplying transformation algorithm, where past Step 1 all positions of the text were still candidates.) If S_j is not empty, then j is a candidate and each k-tuple in S_j enables a different k-linear occurrence at j. The set S_j is represented by a triangulated system of equations whose solutions form S_j.

Time complexity of Step 1. For each position j these systems have identical homogeneous parts. Using Gauss elimination, triangulation of the homogeneous part takes $O(k^3)$ time. We will emulate this triangulation process on the right-hand side of each system. This takes $O(k^2)$ time per position. Step 1 takes a total of $O(k^2 n)$ time.

In Step 2 below, we adapt the duel mechanism for the purpose of invalidating candidacy of positions in the text. Let $j_2 > j_1$ be two positions in the text which are less than m apart (i.e., $j_2 - j_1 < m$). Suppose that following Step 1, j_1 and j_2 are candidates for k-linear transformation occurrences (i.e., neither S_{j_1} nor S_{j_2} is empty).

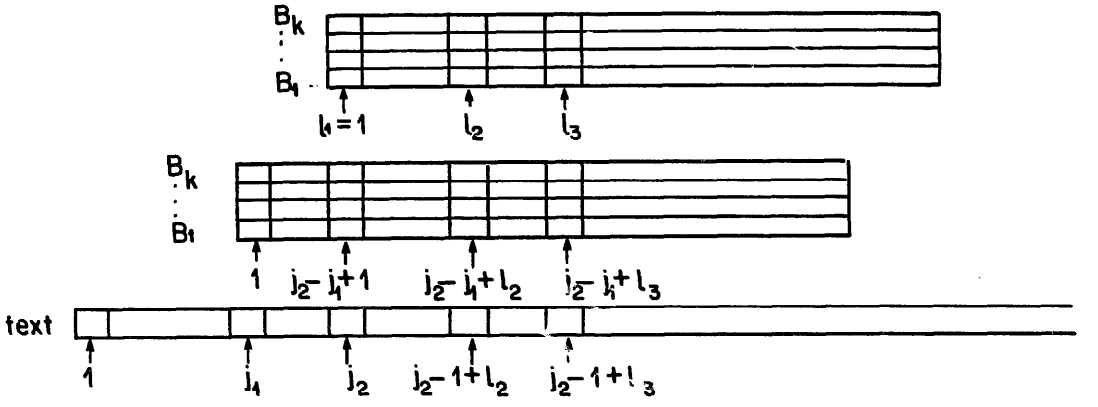

Fig. 7.

A duel between j_1, j_2: Let *witness*(j_2-j_1+1) be $(l_1, \ldots, l_{2k})$ (cf. Fig. 7). We update S_{j_1} by considering the equations of locations $j_2-j_1+l_i$, where $l_i \neq 0$. Specifically, we demand that each k-tuple in S_{j_1} will also satisfy the following equations:

$$\sum_{l=1}^{k} c_{j_1,l} B_{l,j_2-j_1+l_i} = T_{j_2-1+l_i} \quad \forall i,\ 1 \leq i \leq 2k, \text{ where } l_i \neq 0. \tag{3.1}$$

This may result in reduction of the set S_{j_1}. Specifically, more equations will be added to form the new triangulated representation of S_{j_1}. If S_{j_1} becomes empty then the candidacy of j_1 is invalidated.

Similarly, we update S_{j_2}. Specifically, we demand that each k-tuple in S_{j_2} will also satisfy the following equations,

$$\sum_{l=1}^{k} c_{j_2,l} B_{l,l_i} = T_{j_2-1+l_i} \quad \forall i,\ 1 \leq i \leq 2k, \text{ where } l_i \neq 0 \tag{3.2}$$

and if S_{j_2} becomes empty, then the candidacy of j_2 is invalidated. We say that an invalidated candidate is a *loser* in the duel.

Step 2

Following Step 1, we create a (doubly linked) list of candidates (for k-linear transformation occurrence). Each element in this list contains:

(1) the original index j in the text;

(2) the set S_j.

The processing of the list proceeds in substeps, where two elements of the list participate in a substep. Determining the participants in a substep is identical to Step 2 of the text analysis in the previous section. Duels are performed according to the definition above.

Time complexity of Step 2. There are $O(n)$ substeps, in each we may perform a duel. We show that each duel needs $O(k^3)$ time. Thus, Step 2 needs $O(k^3 n)$ time. Consider a duel where j_i is one of the participants. Updating j_i involves adding $O(k)$ equations to S_{j_i} (i.e., to a system in triangular form). This is done in $O(k^3)$ time.

Before we continue to Step 3 we show that Fact 2.2 and Corollary 2.3 remain valid.

3.1. Fact. *Consider the end of Step* 2. *Given two adjacent elements in this list, then there must have been a substep of Step* 2 *in which these two elements participated.*

The proof is similar to the proof of Fact 2.2 and does not involve any new ideas.

3.2. Corollary. *Consider the end of Step* 2. *Let* j_1 *and* j_2 $(j_2 > j_1)$ *be any two adjacent elements in the list. Then, one of the two following conditions holds*:

(1) $j_2 - j_1 \geq m$;

(2) $j_2 - j_1 < m$ *and for each k-tuple* $c_{j_1,1}, \ldots, c_{j_1,k}$ *in* S_{j_1} *and each k-tuple* $c_{j_2,1}, \ldots, c_{j_2,k}$ *in* S_{j_2} *we get*

$$\sum_{l=1}^{k} c_{j_2,l} B_{l,i} = \sum_{l=1}^{k} c_{j_1,l} B_{l,j_2-j_1+i} \quad \forall i,\ 1 < i \leq m-(j_2-j_1).$$

(*In other words,* $c_{j_1,1}, \ldots, c_{j_1,k}, c_{j_2,1}, \ldots, c_{j_2,k}$ *satisfy the equation for any pair of locations of the upper and lower patterns of Fig. 7 which are* $j_2 - j_1$ *apart.*)

Proof. Suppose $j_2 - j_1 < m$. By Fact 3.1, a duel was performed between j_1 and j_2. No one has lost. Each k-tuple $c_{j_1,1}, \ldots, c_{j_1,k}$ in S_{j_1}, is a solution for the system (3.1). Each k-tuple $c_{j_2,1}, \ldots, c_{j_2,k}$ in S_{j_2} is a solution for the system (3.2). Therefore, for each $l_i \neq 0$ in *witness*$(j_2 - j_1 + 1)$ and for each pair of such k-tuples $c_{j_1,1}, \ldots, c_{j_1,k}$ and $c_{j_2,1}, \ldots, c_{j_2,k}$, we get

$$\sum_{l=1}^{k} c_{j_2,l} B_{l,l_i} - \sum_{l=1}^{k} c_{j_1,l} B_{l,j_2-j_1+l_i} = 0.$$

This means that $c_{j_1,1}, \ldots, c_{j_1,k}, c_{j_2,1}, \ldots, c_{j_2,k}$ satisfy each of the equations in $2k$ unknowns suggested by locations $l_i \neq 0$ in *witness*$(j_2 - j_1 + 1)$. The definition of *witness*$(j_2 - j_1 + 1)$ implies that these equations span all the equations suggested by any other location. Thus, $c_{j_1,1}, \ldots, c_{j_1,k}, c_{j_2,1}, \ldots, c_{j_2,k}$ also satisfy each of these equations. This establishes Corollary 3.2. □

We say that any two adjacent elements in the list at the end of Step 2 do *agree.*

3.3. Remark. The transitivity property of the previous section applies also for the k-linear transformation problem. That is, every two elements in the list at the end of Step 2 (not necessarily adjacent) agree.

Step 3

Step 3 is very similar to Step 3 of the previous section. It proceeds in substeps. In substep i, we finish verifying whether a k-linear occurrence starts at the position in the text of element i in the list. Substep 1 deals with a higher index of the text and later steps deal with lower indices. We describe a typical substep i. Let j_i denote the index in the text of element i in list. There are three basic possibilities regarding relevant information from the previous substep.

Possibility 1: the indices j_{i-1} and j_i of the text are less than m apart (i.e., $j_{i-1}-j_i<m$) and there is a k-linear occurrence at index j_{i-1} of the text. Denote by h_i the integer satisfying $j_{i-1}=j_i-1+h_i$ $(h_i\leq m)$. By Corollary 3.2, $T_{j_i-1+r}=\sum_{l=1}^{k} c_{j_i,l}B_{l,r}$ for each $h_i\leq r\leq m$ and each k-tuple, $c_{j_i,1},\ldots,c_{j_i,k}$ in S_{j_i}. It remains to check whether there are k-tuples, $c_{j_i,1},\ldots,c_{j_i,k}$ in S_{j_i} such that $T_{j_i-1+r}=\sum_{l=1}^{k} c_{j_i,l}B_{l,r}$ for each $2\leq r<h_i$. This is done in (at most) h_i-2 iterations for decreasing values of r. In each iteration we add the equation $T_{j_i-1+r}=\sum_{l=1}^{k} c_{j_i,l}B_{l,r}$ to the representation of S_{j_i}. If S_{j_i} becomes empty (i.e., the new system does not have any solution), we conclude that there is no occurrence at index j_i and proceed to substep $i+1$. If, following all h_i-2 iterations, S_{j_i} is not empty, we conclude that each k-tuple in S_{j_i} enables k-linear occurrences at index j_i and proceed to substep $i+1$.

Possibility 2: $j_{i-1}-j_i<m$ and there is no k-linear occurrence at index j_{i-1} of the text. Specifically, substep $i-1$ found $h_{i-1}\leq m$ such that $T_{j_{i-1}-1+h_{i-1}}\neq\sum_{l=1}^{k} c_{j_{i-1},l}B_{l,h_{i-1}}$ for any k-tuple, $c_{j_{i-1},1},\ldots,c_{j_{i-1},k}$ in $S_{j_{i-1}}$.

(a) If $j_{i-1}-1+h_{i-1}\leq j_i-1+m$, then denote by h_i the integer satisfying $j_{i-1}-1+h_{i-1}=j_i-1+h_i$. Clearly, $1<h_i\leq m$. By Corollary 3.2 we get that $T_{j_i-1+h_i}\neq\sum_{l=1}^{k} c_{j_i,l}B_{l,h_i}$ for any k-tuple, $c_{j_i,1},\ldots,c_{j_i,k}$ in S_{j_i}. We conclude that there is no k-linear occurrence at j_i and proceed to substep $i+1$.

(b) If $j_{i-1}-1+h_{i-1}>j_i-1+m$, the integer h_i satisfying $j_{i-1}-1+h_{i-1}=j_i-1+h_i$ is greater than m. We initialize $h_i=m+1$ and apply Possibility 1.

Possibility 3: the indices j_{i-1} and j_i of the text are at least m apart (i.e., $j_{i-1}-j_i\geq m$). Again, we initialize $h_i=m+1$ and apply Possibility 1.

Time complexity of Step 3. In each substep i we move to a new element of the list. In each new iteration of a substep we move to a new position in the text. Upon moving to a new position in the text we add an equation to the representation of S_{j_i} which involves adding the equation to the triangulated form. This needs $O(k^2)$ time per iteration and a total of $O(k^2n)$ time. The number of substeps is $O(n)$ and therefore Step 3 takes $O(k^2n)$ time.

We conclude that the text analysis takes a total of $O(k^3n)$ time. However, if k^2 is larger (in order of magnitude) than m, we can apply a naive approach which will take $O(kmn)$ time.

Implementation Remark. If during the algorithm, the cardinality of the set S_i, with respect to some element i, becomes one, then it will be more efficient to represent the set simply by its member k-tuple and adapt the algorithm to this representation.

3.2. Pattern analysis

The *witness* table is computed using the naive algorithm. We set $l_1=1$. Suppose $l_1,\ldots,l_{i-1}$ were determined. The $i-1$ independent equations are kept as a triangulated system. In order to determine the value of l_i we search locations $j\geq l_{i-1}+1$, in the upper patterns (as illustrated in Fig. 6). Each location j suggests an equation. We add the equation to the system of the $i-1$ independent equations and triangulate the new system. If the added equation is independent of the formers, the new system

has i independent equations and l_i is set to be j. Otherwise, we continue to the next location. If all locations $j \geq l_{i-1}+1$ in the upper patterns do not suggest an ith independent equation, then each of $l_i, \ldots, l_{2k}$ is set to zero. Adding an equation to $O(k)$ independent equations (with $2k$ unknowns) given in triangulated form, and triangulating it needs $O(k^2)$ time. Thus, the computation of *witness* for each j needs $O(k^2 m)$ time. The pattern analysis takes a total of $O(k^2 m^2)$ time.

4. Algorithms for the minimum distance problems

We start this section with an algorithm for finding a minimum distance occurrence of the pattern (without any transformation) in the text. We define the distance between the pattern P and the text starting at position j, denoted L_j, as follows.

$$L_j = \sum_{i=1}^{m} (T_{j-1+i} - P_i)^2 = \sum_{i=1}^{m} T_{j-1+i}^2 + \sum_{i=1}^{m} P_i^2 - 2 \sum_{i=1}^{m} P_i T_{j-1+i}.$$

The algorithm

(1) Compute $\sum_{i=1}^{m} P_i^2$.
(2) Compute $\sum_{i=1}^{m} T_{j-1+i}^2$ $\forall j$, $1 \leq j \leq n-m+1$
(3) Compute the convolution $\sum_{i=1}^{m} P_i T_{j-1+i}$ $\forall j$, $1 \leq j \leq n-m+1$.
(4) Compute L_j $\forall j$, $1 \leq j \leq n-m+1$.
(5) Find a position j for which L_j is minimum.

Complexity. Step (1) takes $O(m)$ time. Each of Steps (2), (4), (5) takes $O(n)$ time. The convolution in Step (3) is computed using the fast Fourier transform in $\min\{O(nm), O(n \log n)\}$ time. The algorithm takes a total of $\min\{O(nm), O(n \log n)\}$ time. ([14] computed convolution in a similar way.)

An algorithm for the minimum distance adding transformation problem

We define the minimum distance between the pattern subject to adding transformation and the text starting at position j as follows. Find, for each position j, a number c_j which provides the minimum in

$$L_j = \sum_{i=1}^{m} (T_{j-1+i} - (P_i + c_j))^2$$

$$= \sum_{i=1}^{m} T_{j-1+i}^2 + \sum_{i=1}^{m} P_i^2 + c_j^2 m + 2c_j \sum_{i=1}^{m} P_i - 2c_j \sum_{i=1}^{m} T_{j-1+i} - 2 \sum_{i=1}^{m} P_i T_{j-1+i}.$$

The minimum distance adding transformation problem is to find a position j and the number c_j for which L_j is a global minimum.

The algorithm

(1) Compute $\sum_{i=1}^{m} P_i^2$ and $\sum_{i=1}^{m} P_i$.
(2) Compute $\sum_{i=1}^{m} T_{j-1+i}^2$ and $\sum_{i=1}^{m} T_{j-1+i}$ $\forall j$, $1 \leq j \leq n-m+1$.

(3) Compute the convolution $\sum_{i=1}^{m} P_i T_{j-1+i}$ $\forall j$, $1 \leq j \leq n-m+1$.

For each position j we get $L_j = c_j^2 m + c_j B_j + D_j$: a quadratic function of c_j (where B_j, D_j are constants that have already been computed). L_j has a minimum because $m > 0$. The adding transformation for which the minimum value of L_j is achieved is $c_j = -B_j/(2m)$.

(4) Compute for each j the minimum value of L_j.

(5) Find a position j for which a global minimum L_j is achieved.

Complexity. Step (1) takes $O(m)$ time. Each of Steps (2), (4), (5) takes $O(n)$ time. Step (3) takes $\min\{O(nm), O(n \log n)\}$ time. The algorithm takes a total of $\min\{O(nm), O(n \log n)\}$ time.

An algorithm for the minimum distance linear transformation problem

We define the minimum distance between the pattern subject to linear transformation and the text starting at position j as follows. Find for each position j, two numbers $c_{j,0}$ and $c_{j,1}$ which provide the minimum in

$$L_j = \sum_{i=1}^{m} (T_{j-1+i} - (c_{j,1} P_i + c_{j,0}))^2$$

$$= \sum_{i=1}^{m} T_{j-1+i}^2 + c_{j,0}^2 m + c_{j,1}^2 \sum_{i=1}^{m} P_i^2 + 2c_{j,0}c_{j,1} \sum_{i=1}^{m} P_i - 2c_{j,0} \sum_{i=1}^{m} T_{j-1+i} - 2c_{j,1} \sum_{i=1}^{m} P_i T_{j-1+i}.$$

The minimum distance linear transformation problem is to find a position j and the numbers $c_{j,0}$ and $c_{j,1}$ for which L_j is a global minimum.

The algorithm. Steps (1), (2), (3), (5) of the algorithm are identical to those in the previous algorithm. Step (4) will be given later.

For each position j we get $L_j = c_{j,0}^2 m + c_{j,1}^2 A_j + c_{j,0} B_j + c_{j,1} D_j + c_{j,0} c_{j,1} E_j + F_j$: a quadratic function of $c_{j,0}$ and $c_{j,1}$ (where A_j, B_j, D_j, E_j, F_j are the constants computed in the steps (1)-(3).

4.1. Claim. *L_j has a minimum.*

The proof of the claim uses the following lemma.

4.2. Lemma. *$m \sum_{i=1}^{m} P_i^2 \geq (\sum_{i=1}^{m} P_i)^2$ and strict inequality holds when not all the pattern elements are equal.*

We omit the proof of the lemma.

Proof of Claim 4.1. A critical point of L_j is a solution of the following system

$$\frac{\partial L_j}{\partial c_{j,0}} = 2c_{j,0} m + 2c_{j,1} \sum_{i=1}^{m} P_i + B_j = 0,$$

$$\frac{\partial L_j}{\partial c_{j,1}} = 2c_{j,0} \sum_{i=1}^{m} P_i + 2c_{j,1} \sum_{i=1}^{m} P_i^2 + D_j = 0.$$

If all the elements of the pattern are equal, then any pattern that can be achieved by linear transformation can also be achieved by only an adding transformation. So, in case where all elements of the pattern are equal, we apply the minimum distance adding transformation algorithm. If the pattern elements are not all equal, Lemma 4.2 implies

$$\frac{m}{\sum_{i=1}^{m} P_i} \neq \frac{\sum_{i=1}^{m} P_i}{\sum_{i=1}^{m} P_i^2}.$$

Hence, the system has only a single solution and L_j has a single critical point. If we show that this critical point is a relative minimum, then it must give a minimum value for L_j and the claim will follow. This point is a relative minimum because the following condition holds (cf. [3, Theorem 2, p. 232]).

$$\frac{\partial^2 L_j}{\partial^2 c_{j,0}} \frac{\partial^2 L_j}{\partial^2 c_{j,1}} - \left(\frac{\partial^2 L_j}{\partial c_{j,0} \partial c_{j,1}}\right)^2 = 4\left[m \sum_{i=1}^{m} P_i^2 - \left(\sum_{i=1}^{m} P_i\right)^2\right] > 0 \quad \text{and}$$

$$\frac{\partial^2 L_j}{\partial^2 c_{j,0}} = m > 0. \quad \square$$

We are now ready for Step 4 of the algorithm.

(4) Compute for each j the minimum value of L_j.

Complexity. Step (4) takes $O(n)$ time. The algorithm takes a total of $\min\{O(nm), O(n \log n)\}$ time.

The problem and algorithm for the minimum distance multiplying transformation are included in the minimum distance linear case.

Considerations given in Section 1 show that the minimum distance k-degree polynomial transformation problem is included in the minimum distance $(k+1)$-linear transformation problem.

An algorithm for the minimum distance k-linear transformation problem

We define the minimum distance between the pattern subject to k-linear transformation and the text starting at position j as follows. For each position j, find k numbers $c_{j,1}, c_{j,2}, \ldots, c_{j,k}$ which provide the minimum in

$$\begin{aligned} L_j &= \sum_{i=1}^{m} \left(T_{j-1+i} - \sum_{l=1}^{k} c_{j,l} B_{l,i}\right)^2 \\ &= \sum_{i=1}^{m} \left[T_{j-1+i}^2 - 2T_{j-1+i} \sum_{l=1}^{k} c_{j,l} B_{l,i} + \left(\sum_{l=1}^{k} c_{j,l} B_{l,i}\right)^2\right] \\ &= \sum_{i=1}^{m} T_{j-1+i}^2 - 2 \sum_{l=1}^{k} c_{j,l} \sum_{i=1}^{m} T_{j-1+i} B_{l,i} \\ &\quad + \sum_{l=1}^{k} c_{j,l}^2 \sum_{i=1}^{m} B_{l,i}^2 + 2 \sum_{l=1}^{k-1} \sum_{r=l+1}^{k} c_{j,l} c_{j,r} \sum_{i=1}^{m} B_{l,i} B_{r,i}. \end{aligned}$$

The minimum distance k-linear transformation problem is to find a position j and numbers $c_{j,1}, \ldots, c_{j,k}$ for which L_j is a global minimum.

The algorithm

(1) Compute $\sum_{i=1}^{m} B_{l,i}B_{r,i}$ $\forall l, r$, $1 \leq l, r \leq k$.
(2) Compute $\sum_{i=1}^{m} T_{j-1+i}^2$ $\forall j$, $1 \leq j \leq n-m+1$.
(3) Compute the k convolutions $\sum_{i=1}^{m} T_{j-1+i}B_{l,i}$ $\forall j$, $1 \leq j \leq n-m+1$ for each l, $1 \leq l \leq k$.

For each j, $1 \leq j \leq n-m+1$, the critical points of L_j are the solutions of the following system

$$\frac{\partial L_j}{\partial c_{j,r}} = -2 \sum_{i=1}^{m} T_{j-1+i}B_{r,i} + 2c_{j,r} \sum_{i=1}^{m} B_{r,i}^2 + 2 \sum_{l \neq r}^{k} c_{j,l} \sum_{i=1}^{m} B_{r,i}B_{l,i} = 0 \quad 1 \leq r \leq k$$

which simplifies to

$$\sum_{l=1}^{k} c_{j,l} \sum_{i=1}^{m} B_{r,i}B_{l,i} = \sum_{i=1}^{m} T_{j-1+i}B_{r,i} \quad 1 \leq r \leq k. \tag{4.1}$$

4.3. Lemma. *If the patterns are linearly independent, then there exists a single solution to system* (4.1).

Proof. We will consider a new $m \times k$ matrix B which is defined as follows. Column i of matrix B is the pattern B_i for $1 \leq i \leq k$. The matrix of the homogeneous part of system (4.1) is actually $B^{\mathrm{T}}B$. When the patterns are all independent, the matrix B is nonsingular. Thus, $B^{\mathrm{T}}B$ is positive definite (see [8, Example 12.14, p. 272]). This implies that the rank of $B^{\mathrm{T}}B$ is k. So the rank of the homogeneous part is k and there exists a single solution to system (4.1). The lemma follows. □

Note that the homogeneous part of (4.1) does not depend on j. Therefore, each system (for each j) has a single solution.

If only $k' < k$ patterns are independent, we apply the minimum distance k'-linear transformation algorithm for k' independent patterns.

Below we assume that the patterns are independent, and Lemma 4.3 implies that there is a single solution to each system. Thus, for each j, L_j has a single critical point. For each j this critical point is a minimum. To see this, we look at matrix A^j which is defined as follows.

$$A_{r,s}^j = \frac{\partial^2 L_j}{\partial c_{j,r}\partial c_{j,s}} = 2 \sum_{i=1}^{m} B_{r,i}B_{s,i}.$$

A^j is equal to $2B^{\mathrm{T}}B$ and is positive definite. For the definition of matrix B and the proof of positive definiteness, see the proof of Lemma 4.3. By [3, Theorem 3,

p. 234], the single critical point for each j is a relative minimum. Note that the matrix A^j does not depend on j.

(4) Compute for each j the minimum value of L_j.

(5) Find a position j for which a global minimum L_j is achieved.

Complexity. Step (1) takes $O(k^2m)$ time. Each of Steps (2), (5) takes $O(n)$ time. Step (3) takes $\min\{O(knm), O(kn \log n)\}$ time. In Step (4) we solve $n-m+1$ systems. These systems have identical homogeneous parts (see (4.1)). Thus, triangulating the homogeneous part is done in $O(k^3)$ time. We will emulate this triangulation process on the right-hand side of each system. This takes $O(k^2)$ time. Step (4) takes $O(k^2n)$ time. The algorithm takes a total of $\min\{O(k(k+m)n), O(k(k+\log n)n)\}$ time.

4.4. Remark. Parallel algorithms for the minimum distance adding transformation problem and the minimum distance linear transformation problem run in $O(\log n)$ time using $O(n)$ processors. In both algorithms the computation of the convolution dominates the running time. It is computed using the parallel fast Fourier transform in $O(\log n)$ time using $O(n)$ processors (see [2, 5]). The parallel algorithm for the minimum distance k-linear transformation problem computes k convolutions. It runs in $O(k \log n)$ time using $O(n)$ processors.

Acknowledgment

We are grateful to Dr. Haim Wolfson for pointing out the reference to Schwartz and Sharir's paper.

References

[1] R.S. Boyer and J.S. Moore, A fast string searching algorithm, *Comm. ACM* **20** (1977) 762–772.

[2] J.W. Cooley and J.W. Tukey, An algorithm for the machine calculations of complex Fourier series, *Math. Comput.* **19** (1965) 297–301.

[3] A. Fridman, *Advanced Calculus* (Holt, Rinehart & Winston, New York, 1971).

[4] Z. Galil and J.I. Seiferas, Time-space-optimal string matching, *J. Comput. System Sci.* **26** (1983) 280–294.

[5] D. Heller, A survey of parallel algorithms in numerical Linear Algebra, *SIAM Rev.* **20** (1978) 740–777.

[6] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977) 323–350.

[7] R.M. Karp and M.O. Rabin, Efficient randomized pattern-matching algorithms, Manuscript, 1980.

[8] S. Lipschitz, *Theory and Problems of Linear Algebra* (McGraw-Hill, New York, 1968).

[9] G.M. Landau and U. Vishkin, Efficient string matching in the presence of errors, in *Proc. 26th IEEE FOCS* (1985) 126–136; this is a preliminary version of [10] and [11].

[10] G.M. Landau and U. Vishkin, Efficient string matching with k mismatches, *Theoret. Comput. Sci.* **43** (1986) 239–249.

[11] G.M. Landau and U. Vishkin, Efficient string matching with k differences, *J. Comput. System Sci*, to appear.

[12] G.M. Landau and U. Vishkin, Introducing efficient parallelism into approximate string matching, in *Proc. 18th ACM Symp. on Theory of Computing* (1986) 220–230; also: *J. Algorithms*, to appear.

[13] D. Sankoff and J.B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison* (Addison-Wesley, Reading, MA, 1983).

[14] J.T. Schwartz and M. Sharir, Identification of partially obscured objects in two and three dimensions by matching of noisy "characteristic curves", *Internat J. Robotics Res.* 6 (1987) 29–44.

[15] U. Vishkin, Optimal parallel pattern matching in strings, *Inform. and Control* 67 (1985) 91–113.

[16] U. Vishkin and M. Yedidia, Manuscript in preparation; also M. Yedida, On transformed pattern matching problems, M.Sc. Thesis, Department of Computer Science, Tel Aviv University, August 1988.